Program Assignment
CPE 357
Systems Programming

CAL POLY
SAN LUIS OBISPO

Assignment 4

Exercise:

Write a high performance program!
In fact, write two programs! Program 2 calls program 1 several times. Program 1 should share the work with its copies over shared memory.

Things to check out:
execv()
"gather" function
named shared memory

Program 1, the performance program:

Download the assignment template and read and understand the code. Check the TODO's.
The program basically multiplies two 10x10 matrices together and prints the result.

The idea is to be able to call this program multiple times at the same time, and each isntance of the program shares the matrices and works ona certain part of the matrix multiplication, so that the computational time becomes a fraction in comparison with only one instance.

The program should be called with two arguments: the par_id and the par_count.
Par_count .. how many instances of this program are there?
Par_id .. which instance are you?

Par_id = 0 (first instance) whould initialize all necessary shared memory pieces with shm_open and the O_CREAT flag and also ftruncate and mmap! All other instances will need to use shm_open() (without the O_CREAT flag) and mmap still!

Write the code for synching! Meaning, no instance should go beyond this point if not all reached this point!

No copy and change the matrix multiplication function, so that different par_id's work on different part of the multiplication. Easiest way: on different rows.

Program 2, the “MPI” program:

This program gets two arguments:
argv[1]: The program which whould be called with execv (name of program1)
argv[2]: How many instances of program one should be called.

For every execv, you need to fork() and call execv in the child, because if successful, execv KILLS the calling process! Call execv for every instance of program 1.

I.e.: if you decide to let program 1 run 4 times in parallel, and program 1 is named “par”, then:
$ ./program2 par 4

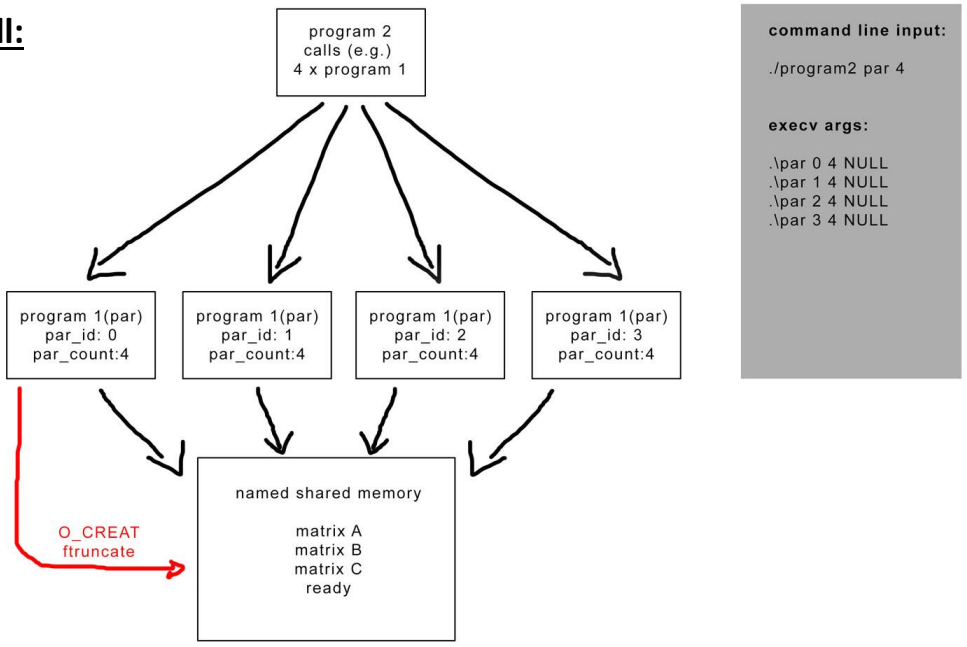
The execv’s need to be called now with:
4 times:

```
execv(programstring,args); with
char *args[4];
and:
args[0] = malloc(100);
args[1] = malloc(100);
… and so on
and:
strcpy(args[0],argv[1]);
strcpy(args[1], X); //X must be par_id
strcpy(args[2], Y); //X must be par_count
NULL //don’t forget for args[4]
```

with these different args:

```
”./par”, ”0”,”4”,NULL
”./par”, ”1”,”4”,NULL
”./par”, ”2”,”4”,NULL
”./par”, ”3”,”4”,NULL
```

In a nutshell:



Compile Flag!

-lrt … for shm_open
#include <sys/mman.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>

What for?

Program 2 is professionally called MPI, Message Passing Interface, and one of the most important high performance software out there (along with OpenMP and CUDA). MPI does the calling of your program, and one has to write the program 1 in respect to it: What if program 1 is called one time? What if 16 times? Etc.

How we test

Checking if you actually did the coding and then let the comaprison function decide.

Submission:

Submit the source code file(s) and the executables: MYNAME_hpc_ass5**.zip**

Bonus?

10% on the assignment if you print the time taken for the multiplication.

More interest?

I need dedicated people to work on a MPI competitor. I offer full elective credits or senior projects. Piece by piece, we could achieve something even better. Where to start: Matrix multiplication is easy, next is matrix inverting and calculating eigenvalues and eigenvectors.